# ebakus

FAST. HIGH CAPACITY. ACCESSIBLE.

## Whitepaper

*We enable Decentralized Applications to reach mass adoption*

# Contents

# Introduction

Until the release of Bitcoin in 2009 designing data integrity sensitive systems solely relied on trusted parties to hold, secure and maintain information. The invention of Bitcoin technology and its successful real world implementation spurred a new wave of innovation and exploration of what can be done with cryptocurrencies and blockchain based technologies.

In 2015 Ethereum was launched further expanding what was possible through bitcoin technology. Ethereum added a Turing complete programming language over the blockchain allowing developers to run decentralized immutable programs, further expanding whats possible. Even though smart contracts have mostly found practical applications in fund-raising through tokens, a lot of exploration is being done in decentralized applications. The concept of a self-governed world computer has never been more close to reality.

# Problem statement

*So if smart contracts give us the technology to start transitioning in decentralized applications, why haven't we seen any successful large scale decentralized Applications yet?*

Bitcoin technology and blockchain based applications are relatively new technologies whereas centralized applications are built on mature technologies. While the benefits of decentralization are profound, in order to realistically transition current centralized services to a decentralized platform and get widespread use, they must perform on par with the competition. **A modern application platform should…**

Be **Scalable**

Decentralized or not, the underlying technology should be able to support millions of requests per second like Google search, Facebook  and others do.

have **Low latency**

Good user experience is not an advantage, nowadays it's a requirement. Having low latency makes a system feel responsive and fast.

allow for **Free transactions**

People are not accustomed to paying fees in order to send and receive data while interacting with a service. Free transactions must be supported.

allow for **Code upgrades and bug recovery**

A development platform should enable Developers to debug and update their Applications easily and reliably, even when they are live.

have **Consistently high performance**

The platform should provide them with high performance for both sequential and parallel algorithms.

All of the above are paramount for developers to achieve pleasant user experiences. Application flows should not have to be designed based on restrictions of the underlying technology. The decentralized and immutable qualities of the platform should be transparent to the user.

# Vision

Decentralized Applications should resemble their centralized counterparts in quality and performance. We designed ebakus to meet modern requirements while building on the technologies developers have already invested in.

**ebakus is fast**

With 1 sec blocks and 2 second finality, it has low latency.

**ebakus has high capacity**

Highly incentivized block providers providing the infrastructure along with minimum block propagation technology allows for high capacity.


**ebakus has free transactions and deployment**

Leveraging a combination of proof of work and proof of stake, ebakus offers fee-less transactions. People with more stake in their accounts enjoy a better quality of service. Developers don’t have to worry about running costs for their decentralized applications.

**ebakus is compatible**

Being compatible to ethereum makes it easy for developers to transfer their work and knowledge to ebakus. Additionally existing developer tools and further infrastructure can be easily reused on ebakus.

**ebakus is frictionless**

ebakus protocol along with the ebakus embeddable wallet makes the existence of blockchain technology transparent to the user and opens dApps for mass adoption.

# Consensus Algorithm

The ebakus blockchain software is utilizing the decentralized consensus algorithm known as *Delegated Proof of Stake (DPOS)*[1]*.* DPOS is currently the only consensus algorithm that can provide the throughput and latency required by modern distributed application running on the blockchain.

## General Concepts

The original proof of work (POW) consensus algorithm proposed by Satoshi and used in the first generation of blockchains like Bitcoin and Ethereum, introduces the main concept that essentially a voting system is formed around the idea that one cpu = one vote. The shortcomings of this approach are easily witnessed in today's POW deployments. Mainly the centralisation and localisation of power on the system. Big mining facilities near cheap electricity and highly popular mining pools are controlling the majority of the hashing power of the network. This power arising from locality (cheep electric power) and need for centralisation to pools (such that smaller power mining makes sense), can not be controlled and be redistributed according to the needs of the majority of the network stake holders. DPOS solves this by assigning this power to the holders of the platform's native token.

In DPOS the blocks are produced by the *block producers,* which are elected through a continuous approval voting system. Block producers are selected among a number of nodes willing to produce blocks called *witnesses*. Anyone can choose to become a witness, but only those that accumulate the most votes acquire the right to produce blocks. The voting power of each stake holder in the network is proportional to their stake -the number of tokens they hold- in the network. This creates a significantly more agile and attack resilient system than POW. A DPOS system can quickly heal in the events of byzantine actors[2] trying to manipulate the network.

## Reducing Latency

The ebakus software by default produces one block every second and at each second only one producer has the authority to produce a block. To achieve this we employ several techniques that allow block propagation to be as efficient as possible.

---

[1] *https://steemit.com/dpos/@dantheman/dpos-consensus-algorithm-this-missing-white-paper*

[2] *https://en.wikipedia.org/wiki/Byzantine_fault_tolerance*

In DPOS what essentially is required for continuous block production at these rates, is the on-time propagation of each block to the next scheduled producer *(handoff)*. If this requirement is met then new blocks can continue to be generated without misses. There is no requirement for the rest of the nodes to be able to keep up and receive the new blocks with any specific minimum latency since the production. By ordering the producer round in a way to minimise the latency of handoff we can achieve the aforementioned target block production rate. Additionally in ebakus each producer produces blocks for 6 consecutive slots before handing it off to the next producer. This further reduces the effects of high latency when the handing off between producers at difference world regions.

Furthermore, each producer has the role of immediately verifying broadcasted blocks. When a block is produced by the authorised producer for the slot and a waiting in turn producer receives it, he will broadcast a verification for the witnessed block. This happens in parallel for all producers. As a result on an ebakus blockchain we have 1 second verification with 1.5-2 second finality even in the worst cases of remote nodes with high latency and slow connections. Compare that to Ethereum that has ~15 seconds verifications and several minutes for finality, and you realise the leap forward in terms of latency and transaction throughput that can be achieved with this architecture. Furthermore we improve the transaction throughput by several orders of magnitude more by exploiting transaction awareness in order to reduce the network serialized block size.

## Improving Throughput

Achieving high transaction rates after reducing the latency of the system to 1 second verification and 2 second finality is essentially bounded by the network connection bandwidth between the producers. In the case of ebakus, with highly incentivised producers, we can expect them to utilise high end hardware and network resources in order to provide the best service for the network. Without the burden of wasting computational power and electricity for POW, all rewards are expected to go to specialized network equipment and computation hardware. This by itself will allow for several megabytes of blocks being trivial to handoff between producers. However the ebakus software is able to reduce the information required to be transmitted for the propagation of a new block to the absolute minimum. We achieve this by utilizing bloom filters and invertible bloom lookup tables to reconcile sets of transactions in blocks with the transaction pool of the producers.

One basic operation of the ebakus software is broadcasting and propagating transactions throughout the network. This process allows transactions from nodes to eventually reach

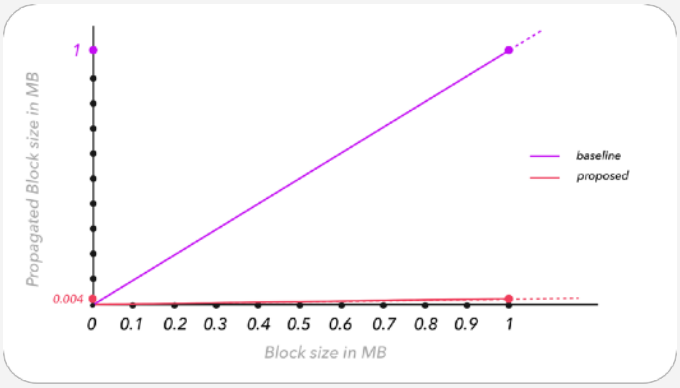
the block producers and be included in one of the next blocks. Given that, it is obvious that at any given time, the block providers are aware of roughly the same transactions. We try to exploit this in order to transmit minimal information between nodes when blocks are propagated. There has been various attempts to achieve this in blockchain technology using IBLTs[3], achieving great reductions in block transmission size. In ebakus software we utilise a combination of bloom filters and IBLTs to achieve unprecedented saving in block transmission sizes[4][5]. Below is an outline of the protocol for block transmission from *node A* to *node B*:

1. B: Requests unknown block and sends the count of transactions in the txpool, $m$
2. A: Sends a bloom filter $S$ and an IBLT $I$ created from the $n$ transactions in the block
3. B: Create IBLT $I'$ from the transactions that pass through $S$. Decodes the subtraction of the two blocks $I \triangle I'$

An IBLT cell is comprised by a count, a hash, a key and a value. For the purpose of block propagation these fields are reduced to the minimum bit length required and we drop the key completely, finally for the value we use part of the transaction id.

In the case the IBLT fails to decode, we retry using a bigger IBLT, and we eventually fallback to normal block propagation, even though it is never necessary under normal conditions.

In the scenario of a transaction pool of 4000 transactions, where the block to be propagated contains half of those, we only need to transmit ~3KB over the network. If we were to transmit only the transaction ids included it would take $2000 \times 32B = 64$ KB. So we transmit just 4.5% of that.




**for Example**

1mb block can fit around 5000 transactions and utilising the proposed solution to propagate them through the network we just need to send approximately 6kb of data.

[3] *https://gist.github.com/gavinandresen/e20c3b5a1d4b97f79ac2*

[4] *David Eppstein, Michael T. Goodrich, Frank Uyeda, and George Varghese. What's the difference?: Efficient Set Reconciliation Without Prior Context. In ACM SIGCOMM, 2011*

[5] *https://people.cs.umass.edu/~gbiss/graphene.pdf*

# Accounts

In ebakus software powered blockchains the account system resembles that of Ethereum with some additions that enable the ebakus specific functionality.

So an ebakus account can be controlled by a private key, or it can be controlled by the code of a smart contract. With a private key controlled account we can send and receive transactions to other accounts to transfer value or send messages. The code controlled accounts can also send and receive messages dictated by the smart contract code. The main difference is that code controlled contracts execute the appropriate code when invoked through a message.

An ebakus account can hold the stake delegated by others, and can delegate stake to other accounts. Staking is always done to the full amount of the coin the account holds. In order to stake part of the coin held, the user should create proxy accounts that will hold the specific coin amount. This can be performed transparently by the wallet software.

Finally, ebakus software provides each account with it's own schema defined database.

# Transactions

### Free Transactions

Blockchains today use fees in order to achieve two main goals. First, to mitigate malicious unsolicited flooding of the network with a huge number of transactions in order to affect the quality of service for normal operations, and use up storage and processing capacity. Second, to incentivise the miners or block producers as they collect those fees.

Adding fees to every transaction greatly hinders the usability of a blockchain. One of our main design goal with ebakus software is to provide free transactions. We do this by solving the two aforementioned problems. We solve the incentive problem by using inflation. The block producers don't depend on the fees of transactions as the software constantly creates new ebakus coins as a reward for them. So now, we have to address the spamming of the network in order to make the inflation truly compensate for the lack of fees additionally to maintaining the quality of service.

We achieve this by utilising an algorithm that uses proof of work in combination with proof of stake. The initial invention of PoW was actually for use in mitigating network denial of service[6]. Most blockchains ended up using fees in order to make it expensive to attack the network, while killing usability and eventually failing to maintain quality of service.

In blockchains running under the ebakus software the network maintains a PoW quantum value. This is the minimum work required by a non stake holder -we will come back to that- for the unit operation on the network. Each operation on the network requires to consume a number PoW quanta in order to be accepted depending on the complexity of computation, the storage requirements, etc. The block providers adjust this PoW quantum to the level that allows normal operation of the network. For example if the network is idle with very little transactions being processed, transactions will require very low PoW to be accepted. In the event someone starts spamming the network with transactions the PoW quantum will be increased so it becomes computationally expensive for him to continue doing so.

However, in order to achieve quality of service for the legitimate users of the network ebakus software does not operate entirely on the global PoW quantum. The global PoW quantum essentially the work quantum required by accounts holding zero amount of ebakus coins. The actual work quantum accepted is adjusted for each account as a function of its stake. Accounts that hold more ebakus coins -and hence have more stake- have a lesser PoW quantum.

The PoW quantum is adjusted in a way that, at anytime, the network resources are allocated proportionally to all the stake holders sending transactions.

On the system we define *Virtual Capacity* to be:

$$c_v = \frac{s_e + s_u}{s_e + s_t}$$

With $s_u$ being the current stake and $s_t$ the total stake in the system. $s_e$ being the epsilon stake in the system.

So if $d_{tr}$ is the transaction supplied difficulty the virtual supplied difficulty is defined as:

---

[6] *A. Back, "Hashcash - a denial of service counter-measure," http://www.hashcash.org/papers/hashcash.pdf, 2002.*

$$d_v = d_{tr} c_v$$

The system then prioritises the transactions according to $d_v$

Ebakus wallets will be able to recalculate the PoW required to send a transaction, so even in cases that the account has zero stake and the network is congested, the user experience will be smooth.

## Selective Evaluation of Transactions

On an ebakus software blockchain there are certain assumptions that are being made about the smart contract execution that allow for more efficient execution parallelism, isolation and scalability. One of the major issues that current blockchain platforms face is the "*everyone executes everything"* problem. ebakus allows for selective evaluation of transactions.

On ebakus we have two kinds of transactions. Those that involve contracts running in the Ethereum compatibility layer, and those involve contracts running in the full ebakus mode. Contracts running in the Ethereum compatibility layer can call other contracts as part of their execution without ever going through the blockchain. This creates a ubiquity as to what part of the global state will be updated by a single transaction. On the other hand contracts running in the native ebakus mode can specify the contracts they are allowed to communicate with directly and therefore affect their state. This way the ebakus software can predict data dependencies and perform calculations in parallel. This allows for better utilisation of hardware and enables great increases in the transaction throughput.

Additionally, this way the ebakus software makes it possible to selectively run only the contracts that interest you. If you run a node only caring about your decentralized exchange DApp, you don't have to execute any other contract. Instead you can selectively execute only the contracts you care for.

## Replay Protection and Blockchain validation

Each transaction on an ebakus software powered blockchain should reference a valid block in the recent past. This reference serves several purposes:

- Provide replay protection as transactions will only be valid on chains that contain that specific block

- Reassure the user that his transaction will only be valid if his chain state was valid at the block he referenced
- Reinforces the fork that users accept as the main line, while cutting off the transactions to any side branch of the chain, that might be malicious or accidental.

# Currency Model

A blockchain based on the ebakus software has storage, network and computational resources provided by the block providers. To optimize availability of these resources and prevent abuse, as we described earlier in the transactions section, we use a combination of proof of work and proof of stake. The amount of proof of work that the sender is required to compute to access the ebakus infrastructure's resources is dynamically calculated to be proportional to the current system load and reversely proportional to ebakus tokens she has staked in her account.

This means that while access to ebakus and its dApps is free for everyone to use, users maintaining a bigger stake in their accounts will have better quality of service when the network is under load. Essentially bigger stake in ebakus tokens translates into priority. Additionally developers won't have to worry about supporting and managing their application costs as they become more popular.

When popular applications emerge, it will result for the load of the network to increase PoW difficulty, therefore users will need to stake more ebakus tokens to maintain high QoS. This will in turn increase token price as demand increases and thus increase the block producers' rewards. As the block producers profits increase they will have to spend more into infrastructure to be more lucrative delegates for the voters to put their stake on.

### Block Reward

A blockchain based on ebakus software will award new tokens to a block producer for every block they produce. The reward will adjust for a 0-5% annual inflation of the total supply, voted upon by the block producers.

# Smart Contracts

A blockchain running with the ebakus software supports Turing complete computations as part of transaction execution, close to what the Ethereum platform supports. The ebakus software does not explicitly dictate, or make any assumptions regarding the language type or form used for this. However, the ebakus software initially supports the

Ethereum Virtual Machine for code execution, both in full Ethereum compatibility mode, and the ebakus enhanced version. This makes the ebakus software based blockchains to have a much lower entry barrier for developers already familiar with the Ethereum platform and the languages Solidity, Viper, LLL, etc.

## Storage Layer

The ebakus software implements two distinct storage layers for smart contracts. These are the following:

1. Ebakus Schema Defined Database layer (EbakusDB)
2. The Ethereum Merkle Tree state storage

One of the major problems a smart contract developer faces when writing DApps for current platforms is that he is constrained to use very basic storage facilities. For example developers of DApps in Ethereum have at their disposal only simple values, arrays and key-value maps. It quickly becomes apparent that any more advanced calculation based on the data stored is difficult to implement, if not impossible due to computational power restrictions. Consider for example a smart contract that has to iterate over a few thousand rows of data to find the top 10 rows in regard to some specific value. This is practically impossible to do on the Ethereum protocol as the gas limit will skyrocket to such levels that it will not be able to fit into a block just by itself. In ebakus software powered blockchains a smart contract is able to perform the above operation on thousands of millions of rows.

## EbakusDB

Each smart contract in ebakus has its own schema defined database (ESDD). This database can support any number of tables with typed fields and indexes. A smart contract is able to perform the following operations on the data:

1. Create/Drop tables
2. Create/Drop indexes on specific fields
3. Retrieve/update/delete single or multiple rows of data
4. Do ordered range queries on these data

The ebakus software makes sure that the data are stored in such a way in order to support the above operations in the most efficient way. The smart contract should not need to implement most common query types by itself.

The EbakusDB layer is providing to the ebakus blockchain a very fast database layer that supports $O(1)$ time and space complexity snapshots. This is essential to the operation of a blockchain system that has requirements for querying old block states. The database achieves high performance by being aware of the transactional log functionality that the layer above it is using and not reimplementing it itself. Therefore achieving ACID compliance without sacrificing performance.

Smart contracts deployed in Ethereum compatibility mode will not be able to make use of the ESDD, hence will not be able to benefit from the extra functionality and performance.

### Native System Smart Contracts

The ebakus software tries to be as modular as possible by implementing most of it's functionality directly in smart contracts. For example the block producer voting and selection is implemented as a smart contract itself. Delegate voting for adjusting core parameters of the network is also implemented in this way. Operating the governance treasury is also running on top of the ebakus core software. These contracts can be chosen to be deployed at the genesis block, or at any other block, and even though they are implemented natively, they make use of the same facilities normal contracts do.

Using the above technique the system will enable the implementation of any different smart contract language or VM. Essentially, the next goal is to support WebAssembly[7] for near native performance and language support.

[7] *http://webassembly.org*

# Governance

In ebakus, software uses a governance framework to make decisions on subjective matters through the process of voting. Our framework was designed to cater for two distinctive categories of issues, health issues and growth issues.

## Dynamic Parameter Configuration

The ebakus software operates on a set of operational parameters i.e. block size, minimum latency, inflation, etc. Even though these parameters were set to cater for the health of the network at launch these parameters may need to change at the future. As the network load increases and block providers earn bigger rewards, they will eventually have to change to improve the speed and capacity of the network. To avoid unnecessary hard forks block providers can vote and dynamically change these parameters.

## Code updates

Another issue that ebakus software allows to vote upon is code change for problematic smart contracts. Smart contracts such as any other piece of code, are prone to bugs and other problems. These should be fixable without having to hard fork.

## Growth

For a blockchain to grow and thrive there is often the need of external resources and services. These can range from marketing resources to budget for projects and services to further grow the ebakus ecosystem. There will be a dynamic parameter that will allow block providers to funnel a percentage of their block reward to fund a reserve designed for this purpose. This will in effect create an internal venture capital arm funded as a part of the inflation introduced by the block providers rewards.

## Voting

Voting for the aforementioned categories is realised by block providers. A proposal for a change is initiated by one of the block providers and the rest follow in voting for its approval. If 17/21 of the block providers vote for the proposal, it will be activated after specific amount of days depending on the type of change. During that activation period, a cancelation vote can be held, where 17/21 votes will result for the proposal to be cancelled.

# Embeddable wallet

Ebakus protocol with its low latency and without any need for a token balance to pay for transaction fees or to be used for staking, sets the foundation for a frictionless experience comparable to its traditional counterparts. To make the most of it, we propose a wallet that can be embedded within the decentralized application and provide the interface to the ebakus protocol without the need for a user to install any third party wallet software as it is commonly required with competing technologies. The moment an ebakus software based application loads it will be ready to interface with the blockchain.

To implement this we propose two solutions.

- **Wallet library embedded inside the dApp**
  In this case the wallet is implemented as part of the dApp and its proposed mostly for native and mobile dApps that want to handle storing the keys themselves and want to completely control the user experience of the wallet interface.

- **Wallet loader JS library**
  For this solution, we or third parties provide a hosted version of the wallet library and the developer loads it through a wallet loader JS library we provide that can be embedded in dApps. The wallet loader library then loads the hosted web wallet through an iframe and provides a secure interface to it. This way users can interact with multiple dApps using the same wallet as it happens with a wallet browser plugin while their keys remain locally stored.

Decentralized applications built on ebakus software, can truly become accessible to mass audiences that are uninitiated to blockchain technology without sacrificing decentralization.

# Conclusions

ebakus was engineered to be a valid technology for a development platform on the blockchain. We designed it based upon proven concepts of blockchain research while keeping it compatible with Ethereum and its development ecosystem. Our goal is to ease in the community to fast, high capacity, governable blockchain technology while keeping

it stable, familiar and frictionless for the users. We aspire to apply solid engineering solutions to a proven and established technology, in order to bridge the gap between DApps and their centralised counterparts. We believe that velocity of innovation is an intrinsic property of itself, and we attempt to invent novel solutions for blockchain technology problems in order to bring it into the future.